

AD-A056 079

GENERAL ELECTRIC CO ARLINGTON VA
PREDICTING PROGRAMMERS' ABILITY TO MODIFY SOFTWARE.(U)
MAY 78 S B SHEPPARD, M A BORST, B CURTIS

F/G 5/9

N00014-77-C-0158

UNCLASSIFIED

TR-78-388100-3

NL

1 OF 1
ADA
056079



END
DATE
FILMED
8 -78
DDC

TR-78-388100-3

AD A 056079

12

2

LEVEL II

TECHNICAL REPORT


PREDICTING PROGRAMMERS' ABILITY TO
MODIFY SOFTWARE

MAY 1978

DDC
RECEIVED
JUL 10 1978
B

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

AD No. _____
DDC FILE COPY

GENERAL  ELECTRIC
INFORMATION SYSTEMS PROGRAMS
ARLINGTON, VIRGINIA

78 06 08 009

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER TR-78-388100 -3	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Predicting Programmers' Ability to Modify Software		5. TYPE OF REPORT & PERIOD COVERED Technical Report
7. AUTHOR(s) S.B. Sheppard, M.A. Borst, B. Curtis & L.T. Love		6. PERFORMING ORG. REPORT NUMBER TR-78-388100-3
9. PERFORMING ORGANIZATION NAME AND ADDRESS General Electric Co. 1755 Jefferson Davis Highway, Arlington, VA 22202		8. CONTRACT OR GRANT NUMBER(s) N00014-77-C-0158
11. CONTROLLING OFFICE NAME AND ADDRESS OFFICE OF NAVAL RESEARCH Arlington, VA 22217		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR 197-037
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE 5/31/78
		13. NUMBER OF PAGES
		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; Distribution unlimited. Reproduction in whole or in part is permitted for any purpose of the U.S. government.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES This research was supported by Engineering Psychology Programs, Office of Naval Research		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Structured Programming Modern Programming Practices Software Complexity Metrics Modification Control Flow Complexity Documentation Software Engineering		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This report describes the second experiment in a program of research designed to identify characteristics of computer software which are related to its psychological complexity. Thirty-six experienced programmers were given unlimited time to make specified modifications to a preliminary program and three experimental programs. The correctness of the modification and the time required to make each modification served as dependent variables. Results indicated that the difficulty of the modification was significantly related to the time to solution. This relationship was described by a hyperbolic function relating		

DD FORM 1473

1 JAN 73

EDITION OF 1 NOV 68 IS OBSOLETE

S/N 0102-014-6601

time to the number of statements to be inserted in the code. Modest effects on the score and time were observed for order of presentation, suggesting a learning effect. On two of the three programs studied, better modifications were made when the control flow of the original programs was well-structured. No performance effects were related to the absence of comments or the type of comments (in-line versus global) used. Moderate relationships with the criteria were observed for several complexity metrics, and these were strongest where metrics were obtained from the modified code rather than the original programs.

ADDITIONAL	
NOTE	100% <input checked="" type="checkbox"/>
NOTE	100% <input type="checkbox"/>
DISTRIBUTION/AVAILABILITY	
BY	
DISTRIBUTION/AVAILABILITY	
Dist.	AVAIL. <input type="checkbox"/> or <input type="checkbox"/>
A	

14

TR-78-388100-3

9

TECHNICAL REPORT

6

PREDICTING PROGRAMMERS'
ABILITY TO MODIFY SOFTWARE.

by

10

S.B. Sheppard, M.A. Borst, B. Curtis,
L.T. Love

11

May 1978

12

38p.

Submitted to: Office of Naval Research
Engineering Psychology Programs
Arlington, Virginia 22217

Contract: 15 N00014-77-C-0158
Work Unit: NR 197-037

General Electric Company
Information Systems Programs
1755 Jefferson Davis Highway, Suite 200
Arlington, Virginia 22202

Approved for public release; distribution unlimited. Reproduction in whole or in part is permitted for any purpose of the United States Government

409 446

78

06

08

009

Job

PREDICTING PROGRAMMERS' ABILITY TO MODIFY
SOFTWARE

by

S.B. Shepaprd, M.A. Borst, B. Curtis &
L.T. Love

Information Systems Programs
General Electric Company
1755 Jefferson Davis Highway, Suite 200
Arlington, Virginia 22202

May 1978

Software Complexity Research Program

Department of Defense (DOD) software production and maintenance is a large, poorly understood, and inefficient process. Recently Frost and Sullivan (The Military Software Market, 1977) estimated the yearly cost for software within DOD to be as large as \$9 billion. DeRoze (1977) has also estimated that 115 major defense systems depend on software for their success. In an effort to find near-term solutions to software related problems, the DOD has begun to support research into the software life-cycle.

A formal 5 year R&D plan (Carlson & DeRoze, 1977) related to the management and control of computer resources was recently written in response to DOD Directive 5000.29. This plan requested research leading to the identification and validation of metrics for software quality. The study described in this paper represents an experimental investigation of such metrics and is part of a larger research program seeking to provide valuable information about the psychological and human resource aspects of the 5 year plan.

The challenge undertaken in this research program is to quantify the psychological complexity of software. It is important to distinguish clearly between the psychological and computational complexity of software. Computational complexity refers to characteristics of algorithms or programs which make their proof of correctness difficult, lengthy, or impossible (Rabin, 1977). For example, as the number of distinct paths through a program increases, the computational complexity also increases. Psychological complexity refers to those characteristics of software which make human understanding of software difficult. No simple relationship between computational and psychological complexity is expected. For example, a program with many control paths may not be psychologically complex, as any regularity to the branching process within a program may be used by a programmer to simplify understanding of the program.

Halstead (1977) has recently developed a theory concerned with the psychological aspects of computer programming. His theory provides objective estimates of the effort and time required to generate a program, the effort required to understand a program, and the number of bugs in a particular program (Fitzsimmons & Love, 1978). Although some predictions of the theory are counter-intuitive and contradict results of previous psychological research, the theory has attracted attention because independent tests of hypotheses derived from it have proven amazingly accurate.

Although predictions of programmer behavior have been particularly impressive, much of the research testing Halstead's theory has been performed without sufficient experimental or statistical controls. Further, much of the data was based on imprecise estimating techniques. Nevertheless, the available evidence has been sufficient to justify a rigorous evaluation of the theory.

Rather than initiate a research program designed specifically to test the theory of software science, a research strategy was chosen which would generate suggestions for improving programmer efficiency regardless of the success of any particular theory. This research focuses on four phases of the software life-cycle: understanding, modification, debugging, and construction. Since different cognitive processes are assumed to predominate in each phase, no single experiment or set of experiments on a particular phase would provide sufficient basis for making broad recommendations for improving programmer efficiency. Each experiment in the series comprising this research program has been designed to test important variables assumed to affect a particular phase of software development. Professional programmers will be used in these experiments to provide the greatest possible external validity for the results (Campbell & Stanley, 1973). In addition, Halstead's theory of software science and other related metrics can be evaluated with these data.

Acknowledgements

The authors gratefully acknowledge the assistance of Dr. Gerald Hahn of General Electric's Corporate Statistics Staff in developing the experimental design. Dr. John O'Hare's careful review of a preliminary version of this report has resulted in substantial improvements.

TABLE OF CONTENTS

	<u>Page</u>
INTRODUCTION	1
METHOD	3
Participants	3
Procedure	3
Experimental Design	3
Independent Variables	4
Programs	4
Complexity of control flow	4
Comments	4
Modifications	6
Covariates	6
Complexity Metrics	6
Halstead's E	6
McCabe's $V(G)$	7
Dependent Variables	8
Analysis	8
RESULTS	10
Pretest	10
Accuracy and Completeness of Modification	10
Time to Complete Modifications	12
Software Complexity Metrics	14
Relationships among metrics	14
Relationships with criteria	14
DISCUSSION	19
REFERENCES	22
APPENDICES	

Predicting Programmers' Ability to Modify Software

Currently, computer programmers spend more of their time modifying existing software or converting it to operate in new environments than in developing new software. Although modifications and maintenance costs have been estimated to be three times higher than those associated with development, managers continue to invest their resources in modification in the mistaken belief that development costs and risks are prohibitive. Thus, modified systems frequently become inefficient collections of concatenated patches. Decisions to modify programs would be aided by systematic information estimating the cost, time, and resources necessary to complete a particular modification. This study was designed to determine the characteristics of software and requested modifications which are related to the speed and accuracy with which programmers are able to make modifications.

Several programming practices should influence the ease with which a program can be modified. Among these practices are documentation and the use of structured coding techniques. Dijkstra (1972) argued that program construction should proceed in a top-down structured fashion, and that programs consistent with these guidelines would be easier to understand, debug, and modify. In an experiment using student programmers, Lucas and Kaplan (1974) found that structured programs took less time to modify. Sheppard, Borst, & Love (1978) found that programs which were structured in a manner to compensate for the lack of suitable control structures in FORTRAN were more easily comprehended by professional programmers.

The use of comments, in-line, global, or both, is another standard software engineering practice which is thought to be related to ease of modification, although there is some contention over how the documentation should be implemented. Global comments preceding a program indicate what objective will be accomplished. In-line comments delineate exactly how and where the objective is fulfilled. Use of in-line comments has been encouraged to simplify the process of making changes to programs (Wilkes, Wheeler, & Gill, 1951, Poole, 1973). Others (Musa, 1976; Shneiderman, 1977) have found that global comments improved student programmers' ability to comprehend and modify programs, but contend that in-line comments seemed distracting. For example, in a FORTRAN modification task with student programmers, Yasukawa (1974) found that a group given global comments performed better than a group given

detailed comments. However, Newsted (1974) found that on short FORTRAN programs, comments preceding the code defining the variables were not useful. Still other computer scientists recommend both global and in-line comments on the theory that too much documentation is impossible.

In parallel with these attempts to improve programmer efficiency, several approaches have been developed for predicting the psychological complexity of software. Presumably, techniques which improve the efficiency of programmers' performance do so by simplifying the cognitive task facing them. Thus, complexity metrics are one way of measuring and validating this assumption. Where such validation is successful these metrics may indicate guidelines for program development.

In 1972, Halstead first published his theory of software physics (renamed software science) stating that algorithms have measurable characteristics analogous to physical laws. According to Halstead (1972a, 1972b, 1975, 1977), the amount of effort (E) required to generate a program can be calculated from simple counts of the actual code. The calculations are based on four quantities from which Halstead derives the number of mental comparisons required to generate a program; the number of distinct operators and operands and the total number of occurrences of operators and operands. Preliminary tests of the theory reported very high correlations (some greater than .90) between Halstead's metric and such dependent measures as the number of bugs in a program (Cornell & Halstead, 1976; Funami & Halstead, 1975), programming time (Gordon & Halstead, 1975), and the quality of programs (Bulut & Halstead, 1974; Elshoff, 1976; Gordon, 1977; Halstead, 1973). A more recent test by Sheppard, Borst, & Love (1978) indicated that the relationship between Halstead's measure and program comprehensibility could be affected by differences among the programs studied.

McCabe (1976) developed a definition of complexity based on the decision structure of the program. McCabe's complexity metric, $V(G)$, is the classical graph-theory cyclomatic number defined as:

$$V(G) = \# \text{ edges} - \# \text{ nodes} + \# \text{ connected components.}$$

Simply stated, McCabe counts the number of elementary control paths through a computer program.

The present study experimentally evaluated the effects of two programming practices (i.e., well structured code and use of comments) on the ease with which a program could be modified. In addition, there was an assessment of the relationship between the speed and accuracy of making a modification and three software complexity metrics, namely Halstead's E , McCabe's $V(G)$, and the total number of statements.

Method

Participants

The sample for this experiment consisted of 36 professional programmers from three different locations within the General Electric Company. These participants had an average of 5.9 years of programming experience ($SD = 4.1$), and all had a working knowledge of FORTRAN. Twenty of these programmers had an engineering background, while the remaining 16 had diverse backgrounds often including statistical and non-numeric programming.

Procedure

A packet of materials was prepared for each participant. The initial instructions to each participant are presented in Appendix A. In a preliminary exercise, participants were asked to modify a short FORTRAN program. All 36 participants were given the same preliminary program and a brief description of its purpose. Participants were given unlimited time to complete the modification. The purpose of this introductory program was two-fold:

- 1) to provide a common basis for comparing the skills of the participants on this type of task, and
- 2) to control for initial learning effects.

Following this initial exercise, participants were presented in turn with the three programs which comprised the experimental task. One modification was requested for each program, and participants were allowed to work at their own pace, taking as much time as needed to implement the modification. An electronic timer was used to record the beginning and ending times of each trial to the nearest minute.

Experimental Design

In order to control for the individual differences in performance, a within-subjects 3^4 factorial design was employed (Hahn and Shapiro, 1966). Three of the programs from a previous experiment in this research program (Sheppard, Borst, & Love, 1978) were used. Three levels of control flow were defined for each of the three programs, and each of these nine versions was presented with three types of documentation for a total of 27 programs. Modifications at three levels of difficulty were developed for each program generating a total of 81 experimental conditions.

Four sets of nine participants were used in the experiment. The participants in the first three sets exhausted the total of 27 program-modification combinations. The fourth set of 9 participants repeated the assignments of one of the three previous sets. Table 1 shows the design for the first 27 participants.

Programmers at each location were randomly assigned to the design so that over the course of their three experimental programs every participant had seen each program, each level of modification, each type of documentation, and each type of control flow. For simplicity the design is presented in Table 1 without regard to the order of presentation to the participants. One of the six possible orders of presentation of the three programs was assigned randomly and without replacement to each participant.

Independent Variables

Programs. Three programs were selected from among those employed in a previous study from this research program (Sheppard, Borst & Love, 1978). The programs were considered to be representative of programs actually encountered by professional programmers. All versions of these three programs were compiled and executed using appropriate test data. The programs were all written in standard FORTRAN.

Complexity of control flow. Three levels of control flow complexity were defined for each program. The least complex level adhered strictly to the tenets of modern structured programming (Dijkstra, 1972). Program flow proceeded from top to bottom with one entry and one exit. Neither backward transfer of control nor arithmetic IFs were allowed.

In FORTRAN awkward constructions often occur when structured programming practices are applied rigorously, such as DO loops with dummy variables (Tenny, 1974). These awkward constructions were largely eliminated in the moderate or quasi-structured level where a more natural control flow was allowed. A judicious use of backward GO TO statements and multiple exits was permitted. IF statements were again restricted to assignment and logical IF's.

In the most complex (i.e., unstructured) versions of each program the control flow was not straightforward. GO TO statements occurred frequently, and backward transfer of control was not restricted. The three-way transfer of control statement (arithmetic IF) was allowed only at this level (Appendix B).

Comments. Three types of comments were tested in this experiment: global, in-line, and none. Global comments provided an overview of the function of the pro-

TABLE 1
EXPERIMENTAL DESIGN

		NONE			GLOBAL			LINE BY LINE			DOCUMENTATION	DIFFICULTY OF MODIFICATION
		1	2	3	1	2	3	1	2	3		
CONTROL FLOW	UNSTRUCTURED	1	23	12	20	15	3	18	2	3		
	SELECT	19	11	7	14	9	25	8	22	17		
	UNIQUE	10	4	27	6	24	13	21	16	5		
QUASI-STRUCTURED	CHISQ	13	8	21	7	27	16	24	10	9		
	SELECT	5	26	15	23	18	4	12	6	20		
	UNIQUE	22	14	2	17	1	19	3	25	11		
STRUCTURED	CHISQ	25	17	6	11	5	22	4	19	14		
	SELECT	16	3	24	2	21	10	27	13	1		
	UNIQUE	9	20	18	26	12	8	15	7	23		

EACH CELL REPRESENTS ONE OF THE THREE ASSIGNMENTS OF A PARTICIPANT WITHIN THE
BLOCK OF 27 PARTICIPANTS

gram and identified the primary variables. In-line comments were interspersed throughout the program and described the specific function of small sections of code. Examples are presented in Appendix C.

Modifications. Three types of modifications were selected for each program as representative of the tasks a programmer might be expected to encounter in relation to such programs. The level of difficulty for seven of the nine modifications increased as the number of lines which had to be added to the code increased. In every case, the hardest modification for each program was the one which required the most lines of code to be inserted into the original program.

Covariates

In order to obtain a measure which was assumed to be related to programming ability, all participants were required to perform the same preliminary task. A short program was given to the participants to modify. Their scores on this task were used as a covariate to measure individual performance differences. Participants were asked their type of programming experience, the number of years they had been programming professionally, and the size of the largest program they had ever constructed. Order of presentation was measured as a situational covariate.

Complexity Metrics

Halstead's E. Halstead's \underline{E} metric was computed from a program (based on Ottenstein, 1976) which had as input the source code listings of nine programs (three separate programs at each of three levels of complexity). The computational formula was:

$$\underline{E} = \frac{(N_1 + N_2) \log_2 (n_1 + n_2)}{(2/n_1) (n_2/N_2)}$$

where,

n_1 = number of unique operators

n_2 = number of unique operands

N_1 = total number of occurrences of operators

N_2 = total number of occurrences of operands

McCabe's $V(G)$. McCabe's metric is the classical graph-theory cyclomatic number, defined as $V(G) = \# \text{ edges} - \# \text{ nodes} + \# \text{ connected components}$. Because the McCabe measure is defined only for programs that adhere strictly to the rules of structured programming, some modifications to the metric were necessary in order to evaluate the less structured control flow versions.

In the simplest program possible, $V(G) = 1$; sequences do not add to the complexity. IF-THEN-ELSE is valued as 2, increasing the complexity by 1. A DO or DO WHILE is also 2, the assumption being that there are really only two control paths, the straight path through the DO and the return to the top, regardless of the number of times executed. Clearly a DO executed 25 times is not 25 times more complex than a DO executed once.

In order to compute the metric for unstructured programs, several alterations were made. An additional RETURN was counted as an extra path in each case, keeping the cyclomatic number the same as that of a "GO TO end". For statements of the form:

IF () 100, 200, 300

the complexity was increased by 2 as opposed to the logical IF, which increases the complexity by 1. These are small changes which appear to be reasonable extensions of McCabe's theory. However, one difficulty arises with the arithmetic IF when two paths are the same:

IF () 100, 100, 200.

In order to standardize the procedure, it was counted in the same way as the standard arithmetic IF, with 2 added to the $V(G)$ metric.

All experimental programs were checked before the experiment to insure that the most complex version of the program had the highest McCabe value and the least complex version had the lowest value.

Dependent Variables

The dependent variables were the correctness of the modification and the time taken by the participant to perform the task. The individual steps necessary for correct implementation of the requested modifications had been delineated in advance and assigned equal weights. The participants' changes were then compared to the criteria. Thus, a percentage score reflecting the correctness of each modification was achieved. All of the responses were scored by the same grader. The time to write a modification was measured to the nearest minute.

Analysis

The analyses of results was conducted in two phases. The first phase was an experimental test of the programming practices, while the second phase was a evaluation of the software complexity metrics.

The first phase, involving experimental manipulations of programming practices, was analyzed by a hierarchical regression analysis. In this analysis domains of variables were entered sequentially into a multiple regression equation to determine if each successive domain significantly improved the prediction of the equation developed from domains already entered. Thus, the order with which domains were entered into the analysis was important. In this study, effects related to differences among participants, programs, modifications, and order were entered into the analysis prior to evaluating the effects of programming practices. The variable domains were entered in the following order:

Differences related to participants and programs

- 1) Pretest scores
- 2) Order of presentation
- 3) Specific program
- 4) Modification

Programming practices

- 5) Program structure
- 6) Documentation

The variables representing the different conditions of domains three through six were effect coded (Kerlinger & Pedhazur, 1973).

The second phase of analysis investigated relationships among Halstead's E, McCabe's V(G), number of statements in the program, and the time and score on the experimental task. Correlations among these measures were examined in both the modified and unmodified programs.

Results

Across all experimental conditions an average score of 62% was received on modifications made ($SD=31\%$). The 108 accuracy scores ranged in value from five scores of 0 to 24 scores of 100; they were negatively skewed. The average time to complete the modifications was 17.9 minutes ($SD=11.4$), ranging from a low of 2 minutes to a high of 59 minutes. The time data were positively skewed. Score and time were uncorrelated.

Pretest

Means and standard deviations for the pretest accuracy score ($M=66\%$, $SD=30\%$) and time to completion ($M=21.4$ min, $SD=14.6$) were similar to those observed on the experimental tasks. Score and time for the pretest were correlated $-.44$ ($df=34$, $p \leq .005$) indicating that participants with high scores worked more quickly; but no causal interpretation is implied. Pretest performance was modestly related to experience in that the number of statements in the largest program a participant had ever written was related to the score ($r_{(34)}=.32$, $p \leq .05$), while participants with more years of experience were able to complete their modifications more quickly ($r_{(34)}=.35$, $p \leq .025$). With the exception of pretest score, none of the individual difference variables were related to the dependent variables on the experimental tasks.

Accuracy and Completeness of Modification

Results presented in Table 2 indicate that overall, only 19% of the variance in scores on the modifications could be predicted by the variable domains measured here. However, there were substantial differences in the degree to which performance on the three programs could be predicted. Performance on two of the programs was reasonably predictable; half of the variance was accounted for in the separate results for each program, and 35% was accounted for in the combined results for both programs. However, the results for a third program were insignificant.

Modest relationships with the performance score were observed for both the pretest and order of presentation. The significance of the order variable suggests the presence of learning or practice effects. However, this interpretation is confounded by the fact that random assignment of presentation order failed to counterbalance the number of times each condition appeared in each position order. With

TABLE 2

Hierarchical Regression Analyses for Accuracy of Modification

Variable domain	ΔR^2	
	All programs ($n=108$)	Two most predictable programs ($n=72$)
1) Pretest score	.05*	.05
2) Order of presentation	.05*	.13**
3) Specific program	.02	.01
4) Modification difficulty	.02	.09**
5) Control flow complexity	.04	.07**
6) Comment type	.01	.00
All domains	.19	.35***

Note: Figures indicate the percent of variance contributed to prediction of performance in addition to that afforded by preceding domains. Significance levels indicate whether this represented a significant contribution to prediction.

* $p \leq .05$
 ** $p \leq .01$
 *** $p \leq .001$

each succeeding experimental task, participants made more complete modifications in less time. However, the two programs on which performance proved most predictable were presented to participants more frequently in the second or third order position.

Accuracy scores differed as a function of the difficulty of the modification on the two most predictable programs. As expected, performance was not as good on modifications which required more lines of code to be inserted. The complexity of the control flow also affected accuracy scores on the two programs for which accuracy was most predictable; modifications to the structured programs were more accurate and complete than those made to unstructured programs.

Accuracy scores did not differ as a function of differences among programs. However, differences among programs moderated relationships among other independent variables and the accuracy criterion. While mean accuracy scores did not differ significantly across programs, relationships between accuracy and other variables did differ among these programs. No differences in scores were observed as a function of the type of comments included in the program.

Time to Complete Modifications

Data presented in Table 3 indicate that 28% of the variance in the time required to complete the modifications across all three programs could be accounted for by variables studied here. The time to complete the modification was more easily predicted than the accuracy of the modification on the program for which prediction of accuracy was low. Although time to completion was not as highly predicted on this program as it was on the other two, including data from it in the regression analysis did not lower the percents of variance accounted for to the extent that had been observed in the accuracy analysis.

Results of the hierarchical regression for time were similar to the results which had been observed for accuracy. The specific program and type of comments were unrelated to the criterion. Unlike the earlier analysis for accuracy scores, however, the pretest results were not related to time to complete the modification. Significant effects were observed for difficulty of the modification and order of presentation, although again, the interpretation of the effect for this latter variable is confounded. Although control flow complexity was significantly related to the accuracy of the modification on the two programs on which accuracy was most predictable, no such effect was observed for the time to complete the modification.

TABLE 3

Hierarchical Regression Analyses for Modification Time

Variable domain	ΔR^2	
	All programs ($n=108$)	Two most predictable programs ($n=72$)
1) Pretest time	.03	.00
2) Order or presentation	.06**	.06
3) Specific program	.01	.01
4) Modification difficulty	.15**	.29**
5) Control flow complexity	.02	.00
6) Comment type	.01	.01
All domains	.28***	.37***

Note: Figures indicate the percent of variance contributed to prediction of performance in addition to that afforded by preceding domains. Significance levels indicate whether this represented a significant contribution to prediction.

** $p \leq .01$
 *** $p \leq .001$

A post hoc inspection of the nine individual modifications in this experiment verified that the number of new statements to be inserted into the code was related to the time required to make the modification. Fitting a hyperbolic function to these data using least squares procedures (Figure 1) resulted in an r^2 of .80 and a standard error of estimate of 2.53. No such relationship was found for score.

Software Complexity Metrics

Relationships among metrics. Correlations among Halstead's and McCabe's metrics and the length (number of statements) are presented in Table 4 for the original programs and their modified versions. There were nine different versions of the original programs (three programs each with three versions of control flow) and 27 modified versions representing three different modifications to each original program. Correlations among these measures were quite high on both the original and modified programs, especially between length and Halstead's E .

Relationships with criteria. Correlations between the three complexity metrics and the two dependent variables are shown in Table 5 for individual datapoints ($n=108$) and data aggregated across the 27 modified programs. In each case, the correlations on the aggregated data were numerically larger than those in the unaggregated data. These larger correlations result from the elimination of individual differences and other sources of error through the aggregation process. The strongest relationship on the original programs was a tendency for higher McCabe values to be associated with lower accuracy scores, but the largest number of significant relationships were observed in relationship to the modified programs. While McCabe's $V(G)$ continued to demonstrate the largest relationship with score, both the length and Halstead's E metrics demonstrated moderate correlations with the time to complete the modification.

Correlations between complexity metrics and performance measures were found to differ with different types of comments. Table 6 presents the correlations between complexity metrics and performance measures for the data generated with each type of comment. All but one of the significant correlations observed occurred when no comments were included in the program. These correlations were stronger on the modified programs than on the original programs.

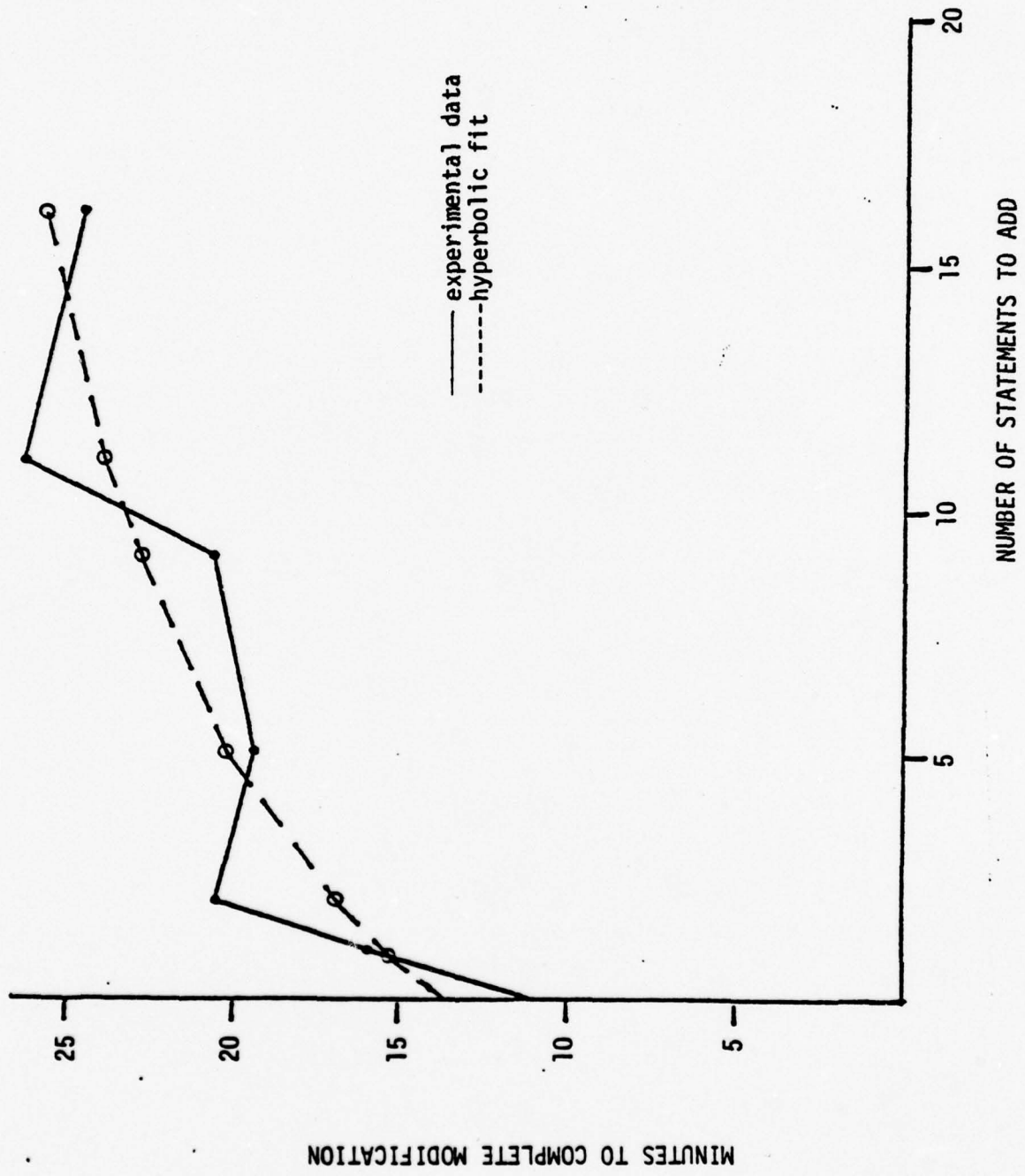


Figure 1. Additional Statements Required Versus Time

TABLE 4
Correlations among Measures of Software Complexity

Metric	Correlations	
	Length	<u>V(G)</u>
Original programs (<u>n</u> =9)		
McCabe's <u>V(G)</u>	.80**	
Halstead's <u>E</u>	.97***	.77**
Modified programs (<u>n</u> =27)		
McCabe's <u>V(G)</u>	.83***	
Halstead's <u>E</u>	.90***	.77***

** $p \leq .01$
*** $p \leq .001$

TABLE 5

Correlations between Complexity Metrics and
Performance Measures for Aggregated
and Unaggregated Data

Criterion	Correlations					
	Original Program			Modified Program		
	Length	V(G)	E	Length	V(G)	E
Accuracy score						
Unaggregated ($n=108$)	-.17*	-.22**	-.12	-.20*	-.22*	-.17*
Aggregated ($n=27$)	-.28	-.38*	-.21	-.37*	-.46**	-.29
Time to completion						
Unaggregated ($n=108$)	.13	.14	.16*	.30***	.23**	.28**
Aggregated ($n=27$)	.20	.22	.25	.45**	.34*	.44**

* $p \leq .05$ ** $p \leq .01$ *** $p \leq .001$

TABLE 6

Correlations between Complexity Metrics and
Performance Measures under Different
Types of Commenting

Criterion and Type of commenting	Correlations					
	Original Program			Modified Program		
	Length	<u>V(G)</u>	<u>E</u>	Length	<u>V(G)</u>	<u>E</u>
Accuracy score						
In line	-.03	-.09	.01	.03	.01	.03
Global	-.14	-.22	-.06	-.23	-.23*	-.18
None	-.31*	-.34*	-.28*	-.37*	-.34*	-.34*
Time to completion						
In line	.14	.09	.16	.16	.07	.16
Global	.02	.13	.09	.18	.21	.21
None	.26	.21	.26	.55***	.42**	.47**

Note: n=36

*p ≤ .05

**p ≤ .01

***p ≤ .001

Discussion

Three aspects involved in the programmers' task of modifying software are: 1) characteristics of the code to be modified, 2) characteristics of the requested modification, and 3) characteristics of the programmer. The main factors found to influence programmers' ability to correctly modify programs were the difficulty of the requested modification and the order of presentation. Other influences were the complexity of the control flow of the original program and individual differences among programmers as measured by a pretest. Each of these factors contributed separately to the prediction of the performance on the task studied. Contrary to expectations, documentation did not influence performance. Several metrics of software complexity were, however, helpful in predicting the accuracy of a modification and the time required to complete it.

It is not surprising that differences in the difficulty of the modifications were related to the time taken to implement the modifications. The effect was more pronounced for time than for accuracy. The number of new lines to be added was the significant criterion for explaining the time spent to finish a task, rather than the number of in-line changes, such as deletions or substitutions of operators and operands. In general, the more new lines to be created, the longer the time expended.

The data reported here suggest that the difficulty of a modification affects the accuracy with which it is implemented. This result agrees with a previous study by Boies and Gould (1974) concerned with syntactic errors. They monitored editor commands which either inserted or substituted code in programs at a large research center. Programs with syntactic errors averaged 32 inserted new lines, while programs without syntactic errors averaged only three inserted lines. There was no difference in the number of substitutions to existing code. These findings probably indicate a greater cognitive difficulty in creating code than in merely deleting or adapting it.

A significant effect due to the order of presentation of the programs suggests the existence of a learning effect as the programmers progressed from task to task. Such effects were not observed in a previous experiment (Sheppard, Borst, & Love, 1978) which involved understanding, as opposed to modifying programs. The failure of random assignment of presentation order to counterbalance the effects of program differences does not permit a clear interpretation of the learning effect.

Control flow complexity was marginally related to the accuracy of the modifi-

cation, but not to the time spent making it. This effect only occurred in the two programs where performance was most easily predicted. Structured code tended to produce more accurate modifications.

It was anticipated that the inclusion of documentation, either global or in-line comments, would significantly improve performance on a modification task. No such improvement occurred on the programs used in this experiment. This is counterintuitive; however, it concurs with the lack of a significant effect from our previous study (Sheppard, Borst, & Love, 1978) for a related cognitive programming aid, mnemonic variable names. This lack of effect for cognitive programming aids may have occurred for one of two reasons. First, in the experiment where levels of variable mnemonicity were manipulated, global comments were provided with all programs. In the current experiment where types of comments were manipulated, mnemonic variable names were provided across all types of comments in all programs. Thus, the existence of one type of cognitive programming aid may have reduced the additional information available from the type of aid being experimentally manipulated, reducing its impact on performance.

A second possibility is that these cognitive programming aids do not contribute significantly to performance for programs of the modular size (approximately 50 lines) employed here. In large systems with many modules and the thousands of lines of code cognitive programming aids may have more impact on performance because of the increased amount of information to be processed. Thus, it may be that program size moderates the relationship between cognitive programming aids such as documentation or mnemonic variable names and performance on various programming tasks.

As expected from previous work (Sheppard, Borst, & Love, 1978), this experiment showed extremely high correlations among the metrics used; length of the program, Halstead's E , and McCabe's $V(G)$. Since Halstead's theory of software science applies primarily to programs in final form rather than programs under development, both the original and modified programs were examined for correlations with the complexity metrics. In every case, there was a higher correlation with performance for complexity metrics computed on the modified programs than had been observed for those computed on the original programs. Nevertheless, the correlations observed are not as high as those reported by Halstead (1977) in other verifications of his theory. The size of the programs employed here may have been

a limiting factor in the results obtained. The range of values for the complexity metrics may not have been sufficient to allow correlational tests to detect the strength of relationships that have been reported in other contexts (Fitzsimmons & Love, 1978). When used with small programs, the metrics were equally productive; when used in a larger system, it may be that one of the metrics will prove superior.

Relating the metrics to performance under conditions distinguished by type of comments uncovered the interesting result that the metrics appeared to predict performance better when there was no documentation presented than when either in-line or global comments appeared. Since the metrics are based on the code, comments of either type add information which the metrics do not evaluate. In this experiment that additional or possibly redundant information neither improved nor hindered the programmer's performance in any way. This suggests that comments are not ignored but that their effect on the programming tasks may be more complex than can be explained by simple effects on performance.

Differences among programs played an important role in this experiment, as they had done in a previous study (Sheppard, Borst, & Love, 1978). The complexity metrics provided one source of information about program differences, but there were other factors within the programs which were not assessed by these metrics. Such factors may involve the complexity of the task performed by the program. Thus, the cognitive difficulty of the program to the programmer may involve an interaction between program characteristics and individual differences, such as the programmer's experience with similar programs. Further research on complexity metrics should evaluate ways of assessing the complexity of the task performed by the program, in addition to the factors currently assessed by the Halstead and McCabe metrics.

References

- Boies, S.J., & Gould, J.D. Syntactic errors in computer programming. Human Factors, 1974, 16, 253-257.
- Bulut, N., & Halstead, M.H. Impurities found in algorithm implementation (Tech. Rep. CSD-TR-111). West Lafayette, IN : Purdue University, Computer Science Department, 1974.
- Campbell, D., & Stanley, J.C. Experimental and quasi-experimental design for research. Chicago: Rand McNally, 1966.
- Carlson, W.E., & DeRoze, B. Defense system software research and development plan. Unpublished manuscript, Arlington, VA : Defense Advanced Research Projects Agency, September 1977.
- Cornell, L., & Halstead, M.H. Predicting the number of bugs expected in a program module (CSD-TR-205). West Lafayette, IN : Purdue University, Computer Science Department, October 1976.
- DeRoze, B. Software research and development technology in the Department of Defense. Paper presented at the AIIE Conference on Software, Washington, D.C.: December 1977.
- Dijkstra, E.W. Notes on structured programming. In O.J. Dahl, E.W. Dijkstra, & C.A.R. Hoare (Eds.) Structured programming. New York: Academic Press, 1972.
- Elshoff, J.L. Measuring commercial PL/I programs using Halstead's criteria. SIGPLAN Notices, 1976, 11, 38-46.
- Fitzsimmons, A.B., & Love, L.T. A review and evaluation of software science. ACM Computing Surveys, 1978, 10, 3-18.
- Funami, Y., & Halstead, M.H. A software physics analysis of Akiyama's debugging data (Tech. Rep. CSD-TR-144). West Lafayette, IN : Purdue University, Computer Science Department, May 1975.
- Gordon, R.D. A measure of mental effort related to program clarity. Unpublished doctoral dissertation, Purdue University, 1977.
- Gordon, R.D., & Halstead, M.H. An experiment comparing FORTRAN programming times with the software physics hypothesis (Tech. Rep. CSD-TR-167). West Lafayette, IN : Purdue University, Computer Science Department, 1975.
- Hahn, G. J., & Shapiro, S.S. A catalogue and computer program for the design and analysis of orthogonal symmetric and asymmetric fractional factorial experiments (Tech. Rep. 66-C-165). Schenectady, NY: General Electric, May 1966.
- Halstead, M.H. Natural laws controlling algorithm structure. SIGPLAN Notices, 1972, 7, 2. (a)

- Halstead, M.H. A theoretical relationship between mental work and machine language programming. (Tech. Rep. CSD-TR-67). West Lafayette, IN: Purdue University, Computer Science Department, May 1972.(b)
- Halstead, M.H. An experimental determination of the "purity" of a trivial algorithm (Tech. Rep. CSD-TR-73). West Lafayette, IN: Purdue University, Computer Science Department, 1973.
- Halstead, M.H. Software physics: Basic principles (Tech. Rep. RJ-1582). Yorktown Heights, NY: IBM, 1975.
- Halstead, M.H. Elements of software science. New York: Elsevier North-Holland, 1977.
- Kerlinger, F.N., & Pedhazur, E.J. Multiple regression in behavioral research. New York: Hold, Rinehart & Winston, 1973.
- Lucas, H.C., Jr., & Kaplan, R.B. A structured programming experiment. The Computer Journal, 1974, 19, 136-138.
- McCabe, T.J. A complexity measure. IEEE Transactions on Software Engineering, 1976, SE-2, 308-320.
- Musa, J.D. An exploratory experiment with "foreign" debugging of programs. In Proceedings of the symposium on computer software engineering. New York: Polytechnic Institute of New York, 1976.
- Newsted, P.R. FORTRAN program comprehension as a function of documentation. Program Information Abstracts: Second Annual Computer Science Conference. Detroit, MI: 1974.
- Ottenstein, K.J. A program to count operators and operands for ANSI-FORTRAN modules. (Tech. Rep. CSD-TR-196). West Lafayette, IN: Purdue University, Computer Science Department, June 1976.
- Poole, P.C. Debugging and testing. In F.L. Bauer (eds) Advanced course in software engineering. New York: Springer-Verlag, 1973.
- Rabin, M.O. Complexity of computations. Communications of the ACM, 1977, 20, 625-633.
- Sheppard, S.B., Borst, M.A., & Love, L.T. Predicting software comprehensibility (Tech. Rep. TR-78-388100-2). Arlington, VA: General Electric/ISP, 1978.
- Snheiderman, B. Measured computer program quality and comprehension. International Journal of Man-Machine Studies, 1977, 9, 465-478.
- Tenny, T. Structured programming in FORTRAN. Datamation, 1972, 20, 110-115.
- The Military Software Market (Rep. 427). New York: Frost & Sullivan, 1977.

Wilkes, M.V., Wheeler, D.J., & Gill, S. The preparation of programs for an electronic digital computer. Reading, MA : Addison-Wesley, 1951.

Yasukawa, K. The effect of comments on program understandability and error correction. Unpublished paper, 1974.

APPENDIX A
INSTRUCTIONS TO PARTICIPANT

GOOD MORNING!!!

Today we are going to ask you to participate in an experiment which we hope will be both entertaining and challenging.

This work, sponsored by the Office of Naval Research, is being done to make computer programs easier to modify. To do this, we will give you three separate programs and modification slips and ask you to make the required modification to each program.

Our purpose is to evaluate characteristics of programs which make them easier to modify. It is not to evaluate computer programmers. Your performance on a program will be compared only to your performance on other programs. Your only competition is yourself. All programs and papers that you will be handed are carefully numbered so it is not necessary for you to put your name on any of these.

We would like you to answer the following questions for our research purposes:

1. How long have you been programming in FORTRAN professionally?

_____ years _____ months

2. Please circle one of the following: Has your primary experience been with Engineering, Statistical, Non-Numeric, or Other programs?

Also, please briefly describe your specific areas of programming experience.

3. Approximately how many source code instructions were in the longest FORTRAN program that you have ever written? Please exclude blank lines and comments

_____.

During this experiment, each of you will be working on a different program. If someone else seems to finish earlier than you, don't be concerned. They will have been working on something else entirely which might not require as much time.

We will begin this morning with a short test program. We will ask you to make the modification as accurately as possible, and to raise your hand as soon as you are through. Because of the concentration required for this task, we ask you to make an extra effort to remain quiet so that others will not be distracted.

If there are any questions, please ask them at this time.

APPENDIX B

CONTROL STRUCTURES ALLOWED IN THE THREE LEVELS OF COMPLEXITY

STRUCTURED	QUASI-STRUCTURED	UNSTRUCTURED
DO LOOP FORWARD EXIT ONLY	DO LOOP BACKWARD EXIT ALLOWED	EXPANDED DO LOOP
LOGICAL IF		ARITHMETIC IF IF () -, 0, +
GO TO FORWARD ONLY	GO TO BACKWARD ALLOWED	
COMPUTED GO TO FORWARD ONLY - SINGLE ENTRY AND EXIT	COMPUTED GO TO BACKWARD ALLOWED	
SINGLE RETURN	MULTIPLE RETURNS ALLOWED	ASSIGNED GO TO

APPENDIX C

SELECT, UNSTRUCTURED, NO COMMENTS

```

SUBROUTINE SELECT(N,STR,ERR,N1,N2)
EXTERNAL RND
INTEGER STR(N),MIX(26),RCHR,ERR,STO,ALP(26)
DATA ALP/1HA,1HB,1HC,1HD,1HE,1HF,1HG,1HH,1HI,1HJ,
1 1HK,1HL,1HM,1HN,1HO,1HP,1HQ,1HR,1HS,1HT,1HU,1HV,1HW,
2 1HX,1HY,1HZ/
IF(N-25) 90,90,70
70 ERR=99
GO TO 500
90 ERR=0
I = 0
92 I = I + 1
IF (I - 26) 95, 95, 100
95 MIX(I)=ALP(I)
GO TO 92
100 I = 0
102 I = I + 1
IF (I - 25) 115, 115, 120
115 RCHR=RND(1,26,N1,N2)
STO=MIX(RCHR)
MIX(RCHR)=MIX(I)
MIX(I)=STO
GO TO 102
120 I = 0
122 I = I + 1
IF (I - N) 125, 125, 500
125 STR(I)=MIX(I)
GO TO 122
500 RETURN
END
FUNCTION RND(L1, L2, N1, N2)
INTEGER L1, L2, N1, N2
IF (L2-L1) 10,20,20
10 INDP = L2
L2 = L1
L1 = INDP
20 SIZ = L2 - L1 + 1
R = RAN (N1, N2)
RND = IFIX (R * SIZ + FLOAT (L1))
END

```

UNIQUE, STRUCTURED, GLOBAL COMMENTS

SUBROUTINE UNIQUE

PURPOSE

TO STRIP ADJACENT DUPLICATE ITEMS;
FOR EXAMPLE, TO ELIMINATE DUPLICATES IN A PRE-SORTED MAILING
LIST, OR HOMOGRAPHS (WORDS WITH THE SAME SPELLING BUT
DIFFERENT MEANINGS) FROM A DICTIONARY.

USAGE

CALL UNIQUE(ARR, N, MD

DESCRIPTION OF PARAMETERS

ARR - ARRAY OF ITEMS
N - NUMBER OF ITEMS IN ARRAY
M - MAXIMUM LENGTH OF AN ITEM
ALT1 - BUFFER FOR ALTERNATE ITEMS
ALT2 - BUFFER FOR ALTERNATE ITEMS
I1 - ITEM NUMBER IN ORIGINAL LIST
I2 - ITEM NUMBER IN STRIPPED LIST

FUNCTION SUBPROGRAMS REQUIRED

NONE

```
SUBROUTINE UNIQUE(ARR,N,MD
INTEGER ALT1(MD),ALT2(MD),ARR(N,MD
I1=1
I2=1
DO 10 L=1,M
  ALT1(L)=ARR(I1,L)
10 CONTINUE
DO 90 LP=1,N
DO 20 L=1,M
  ARR(I2,L)=ALT1(L)
20 CONTINUE
  I2=I2+1
DO 40 KTR=1,N
  I1=I1+1
  IF (I1 .GT. N) GO TO 100
DO 30 L=1,M
  ALT2(L)=ARR(I1,L)
30 CONTINUE
DO 40 L=1,M
  IF (ALT1(L) .NE. ALT2(L)) GO TO 50
40 CONTINUE
DO 60 L=1,M
  ARR(I2,L)=ALT2(L)
50 CONTINUE
  I2=I2+1
DO 80 KTR=1,N
  I1=I1+1
  IF (I1 .GT. N) GO TO 100
DO 70 L=1,M
  ALT1(L)=ARR(I1,L)
70 CONTINUE
DO 80 L=1,M
  IF (ALT1(L) .NE. ALT2(L)) GO TO 90
80 CONTINUE
90 CONTINUE
100 N=N-1
RETURN
END
```

CHISQ, QUASI-STRUCTURED, IN-LINE COMMENTS

```

C  CALCULATES DEGREES OF FREEDOM AND CHI-SQUARE
C  FOR A GIVEN CONTINGENCY TABLE OF OBSERVED FREQUENCIES
  SUBROUTINE CHISQ(MAT,N,M,CS,DEG,ERR,RTOT,CTOT)
    INTEGER ERR,DEG,PTR
    REAL MAT
    DIMENSION MAT(100),RTOT(N),CTOT(M)
C  MAXIMUM NUMBER OF CELLS ALLOWED IS 100
C  # OF CELLS = # OF ROWS * # OF COLUMNS
    NM=N*M
    ERR=0
    CS=0.0
C  FIND DEGREES OF FREEDOM
    DEG=(N-1)*(M-1)
    IF (DEG .GT. 0) GO TO 10
C  NUMBER OF DEGREES OF FREEDOM IS ZERO
    ERR=2
    RETURN
C  COMPUTE TOTALS OF ROWS
10  DO 20 I=1,N
      RTOT(I)=0.0
      PTR=I-N
      DO 20 J=1,M
        PTR=PTR+N
        RTOT(I)=RTOT(I)+MAT(PTR)
      20  CONTINUE
C  COMPUTE TOTALS OF COLUMNS
C  PTR POINTS TO CELL IN ARRAY
      PTR=0
      DO 30 J=1,M
        CTOT(J)=0.0
        DO 30 I=1,N
          PTR=PTR+1
          CTOT(J)=CTOT(J)+MAT(PTR)
        30  CONTINUE
C  COMPUTE GRAND TOTAL
      GTOT=0.0
      DO 40 I=1,N
        GTOT=GTOT+RTOT(I)
      40  CONTINUE
C  COMPUTE CHI SQUARE
      PTR=0
      DO 50 J=1,M
        DO 50 I=1,N
          PTR=PTR+1
          EXPT=RTOT(I)*CTOT(J)/GTOT
C  IS EXPECTED VALUE LESS THAN 1?
          IF (EXPT .LT. 1.0) ERR=1
          CS=CS+(MAT(PTR)-EXPT)*(MAT(PTR)-EXPT)/EXPT
        50  CONTINUE
      IF (NM .NE. 4) GO TO 70
C  COMPUTE CHI SQUARE FOR 2 BY 2 TABLE (SPECIAL CASE)
      60  CS=GTOT*(ABS(MAT(1)*MAT(4)-MAT(2)*MAT(3))-GTOT/2.0)**2
          1 / (CTOT(1)*CTOT(2)*RTOT(1)*RTOT(2))
      70  RETURN
      END

```

OFFICE OF NAVAL RESEARCH, CODE 455
TECHNICAL REPORTS DISTRIBUTION LIST

Director, Engineering Psychology
Programs, Code 455
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217 (5 cys)

Col. Henry L. Taylor, USAF
OAD (E&LS) ODDRE&E
Pentagon, Room 3D129
Washington, D.C. 20301

Director, Statistics and Probability
Program, Code 436
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Commanding Officer
ONR Branch Office
Attn: Dr. J. Lester
Building 114, Section D
666 Summer Street
Boston, MA 02210

Dr. Bruce McDonald
Office of Naval Research
Scientific Liaison Group
American Embassy, Room A-407
APO San Francisco, CA 96503

Naval Research Laboratory
Attn: Code 5707
Washington, D.C. 20375

Office of the Chief of Naval
Operations, OP987H
Personnel Logistics Plans
Department of the Navy
Washington, D.C. 20350

Commander
Naval Air Systems Command
Human Factors Programs, AIR 340F
Washington, D.C. 20361

Mr. T. Momiyama
Naval Air Systems Command
Advance Concepts Division, AIR03P34
Washington, D.C. 20361

Defense Documentation Center
Cameron Station
Alexandria, VA 22314 (12 cys)

Dr. Stephen J. Andriole
Acting Director, Cybernetics Technology
Office
Advanced Research Projects Agency
1400 Wilson Blvd.
Arlington, VA 22209

Director, Information Systems Program,
Code 437
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Commanding Officer
ONR Branch Office
Attn: Dr. Charles Davis
536 South Clark Street
Chicago, IL 60605

Commanding Officer
ONR Branch Office
Attn: Dr. E. Gloye
1030 East Green Street
Pasadena, CA 91106

Director, Naval Research Laboratory
Technical Information Division
Code 2627
Washington, D.C. 20375 (6 cys)

Mr. Arnold Rubinstein
Naval Material Command
NAVMAT 08T24
Department of the Navy
Washington, D.C. 20360

Commander
Naval Air Systems Command
Crew Station Design, AIR 5313
Washington, D.C. 20361

Commander
Naval Electronic Systems Command
Human Factors Engineering Branch
Code 4701
Washington, D.C. 20360

Mr. James Jenkins
Naval Sea Systems Command
Code 06H1-3
Washington, D.C. 20362

Director
Behavioral Sciences Department
Naval Medical Research Institute
Bethesda, MD 20014

Chief, Aerospace Psychology Division
Naval Aerospace Medical Institute
Pensacola, FL 32512

Bureau of Naval Personnel
Special Assistant for Research
Liaison
PERS-OR
Washington, D.C. 20370

Dr. Fred Muckler
Navy Personnel Research and Development
Center
Manned Systems Design, Code 311
San Diego, CA 92152

LCDR P.M. Curran
Human Factors Engineering Branch
Crew Systems Department Code 4021
Naval Air Development Center
Johnsville
Warminster, PA 18950

Human Factors Section
Systems Engineering Test Directorate
U.S. Naval Air Test Center
Patuxent River, MD 20670

Human Factors Engineering Branch
Naval Ship Research and Development
Center, Annapolis Division
Annapolis, MD 21402

Human Factors Department
N215
Naval Training Equipment Center
Orlando, FL 32813

Dr. Gary Poock
Operations Research Department
Naval Postgraduate School
Monterey, CA 93940

Dr. James Curtin
Naval Sea Systems Command
Personnel & Training Analyses Office
NAVSEA 074C1
Washington, D.C. 20362

Dr. George Moeller
Human Factors Engineering Branch
Submarine Medical Research Laboratory
Naval Submarine Base
Groton, CT 06340

Mr. Phillip Andrews
Naval Sea Systems Command
NAVSEA 0341
Washington, D.C. 20362

Navy Personnel Research and Development
Center
Management Support Department
Code 210
San Diego, CA 92152

Navy Personnel Research and Development
Center
Code 305
San Diego, CA 92152

LCDR William Moroney
Human Factors Engineering Branch
Code 1226
Pacific Missile Test Center
Point Mugu, CA 93042

Dr. John Silva
Man-System Interaction Division
Code 823, Naval Ocean Systems Center
San Diego, CA 92152

Naval Training Equipment Center
Attn: Technical Library
Orlando, FL 32813

Dr. Alfred F. Smode
Training Analyses and Evaluation Group
Naval Training Equipment Center
Code N-00T
Orlando, FL 32813

Dr. A.L. Slafkosky
Scientific Advisor
Commandant of the Marine Corps
Code RD-1
Washington, D.C. 20380

Mr. J. Barber
Headquarters, Department of the Army
Army, DAPE-PBR
Washington, D.C. 20546

Dr. Edgar M. Johnson
Organization and Systems Research
Laboratory
U.S. Army Research Lab
5001 Eisenhower Avenue
Alexandria, VA 22333

U.S. Air Force Office of Scientific
Research
Life Sciences Directorate, NL
Bolling Air Force Base
Washington, D.C. 20332

Lt. Col. Joseph A. Birt
Human Engineering Division
Aerospace Medical Research Lab.
Wright Patterson AFB, OH 45433

Dr. Robert Williges
Human Factors Laboratory
Virginia Polytechnic Institute
130 Whittemore Hall
Blacksburg, VA 24061

Dr. Gershon Weltman
Perceptrics, Inc.
6271 Variel Avenue
Woodland Hills, CA 91364

Dr. H. Rudy Ramsey
Science Applications, Inc.
40 Denver Technological Center West
7835 East Prentice Avenue
Englewood, CO 80110

Dr. Jesse Orlansky
Institute for Defense Analyses
400 Army-Navy Drive
Arlington, VA 22202

Dr. Stanley Deutsch
Office of Life Sciences
HQS, NASA
600 Independence Avenue
Washington, D.C. 20546

Dr. Joseph Zeidner
Acting Technical Director
U.S. Army Research Institute
500 Eisenhower Avenue
Alexandria, VA 22333

Technical Director
U.S. Army Human Engineering Labs
Aberdeen Proving Ground
Aberdeen, MD 21005

U.S. Army Aeromedical Research Lab.
Attn: CPR Gerald P. Krueger
Ft. Rucker, AL 36362

Dr. Donald A. Tompiller
Chief, Systems Engineering Branch
Human Engineering Division
USAF AMRL/HES
Wright-Patterson AFB, OH 45433

Air University Library
Maxwell Air Force Base, AL 36112

Dr. Arthur I. Siegel
Applied Psychological Services, Inc.
404 East Lancaster Street
Wayne, PA 19087

Dr. Robert R. Mackie
Human Factors Research, Inc.
Santa Barbara Research Park
6780 Cortona Drive
Goleta, CA 93017

Dr. Meredith Crawford
5606 Montgomery Street
Chevy Chase, MD 20015

Dr. Robert G. Pachella
University of Michigan
Department of Psychology
Human Performance Center
330 Packard Road
Ann Arbor, MI 48104

Director, National Security Agency
Attn: Dr. Douglas Cope
Code R51
Ft. George G. Meade, MD 20755

Journal Supplement Abstract Service
American Psychological Association
1200 17th Street, NW
Washington, D.C. 20036 (3 cys)

Director, Human Factors Wing
Defense & Civil Institute of
Environmental Medicine
Post Office Box 2000
Downsville, Toronto, Ontario
CANADA

Naval Electronics Laboratory Center
Advanced Software Technology Div.
Code 5200
San Diego, CA 92152

Mr. Kin B. Thompson
Technical Director
Information Systems Division
OP-91T
Office of the Chief of Naval Operations
Washington, D.C. 20350

Stephen Fickas
NOSC
San Diego, CA 92152

Dr. William A. McClelland
Human Resources Research Office
300 N. Washington Street
Alexandria, VA 22314

Dr. A.D. Baddeley
Director, Applied Psychology Unit
Medical Research Council
15 Chaucer Rd.
Cambridge, CB2 2EF
ENGLAND

Capt. Grace M. Hopper
NAICOM/MIS Planning Board
OP-916D
Office of the Chief of Naval Operations
Washington, D.C. 20350

Advanced Research Projects Agency
Information Processing Techniques
1400 Wilson Blvd.
Arlington, VA 22209

Computer & Information Science Lab Center
Ohio State University
Columbus, OH 43210